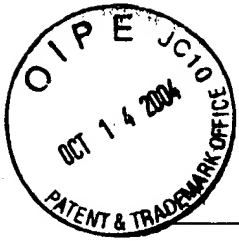


10-18-04

AF/2122
EPW



**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

In re Application of: Stephen E. Fischer

Serial No.: 09/757,517

Filing Date: January 10, 2001

For: DEPENDENCY SPECIFICATION
USING TARGET PATTERNS

Group Art Unit: 2122

Examiner: Andre R. Fowlkes

Date: October 14, 2004

Mail Stop Appeal Brief - Patents
Commissioner of Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

EXPRESS MAIL" mailing number EV453780352US

Date of Deposit: October 14, 2004, I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Mail Stop Appeal Brief - Patents, Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Name: Carol M. Thomas Date: October 14, 2004

Signature

APPEAL BRIEF

This is an appeal from the Final Rejection of the Examiner mailed May 19, 2004, rejecting all the claims in the application, to wit, claims 1-7 and 9-22 and 24-26. A Notice of Appeal and the Appeal Fee, along with a Petition for one-month extension of time were mailed to the United States Patent and Trademark Office on September 2, 2004 and received by the United States Patent and Trademark Office on September 7, 2004.

The foregoing Appeal Brief is prepared in accordance with the new revised requirements of 37 C.F.R. § 1.192. Accordingly, one copy of the brief is enclosed. Please charge the Appeal fee of \$340 for this brief and any over or under payment to Deposit Account No. 09-0458, of the assignee International Business Machine Corporation.

REAL PARTY IN INTEREST

The real party in interest is the assignee of all rights in this application, International Business Machines Corporation, a corporation of the State of New York, having a place of business at Armonk, New York 10504.

RELATED APPEALS AND INTERFERENCES

There are no other prior or pending appeals, interferences or judicial proceedings known to Appellant, Appellant's legal representatives, or assignee, which may be related to, directly affect or be affected by, or have a bearing on the Board's decision on this appeal.

STATUS OF CLAIMS

The subject application was filed on January 10, 2001 with claims 1-15. A non-final Office Action was mailed December 23, 2003 in which the Examiner had rejected all the claims in the application, to wit, claims 1-15. An Amendment responsive to the non-final Office Action was mailed March 23, 2004, along with a Replacement Amendment mailed March 30, 2004 wherein a reference date was merely corrected for. In such

Amendments formal drawings were submitted replacing the informal drawings, informalities were corrected for in the specification, and claims 1-7 and 9-15 were amended, claim 8 canceled, and claims 16-25 added.

A Final Rejection Office Action rejecting all the pending claims in the application, to wit, claims 1-7 and 9-25, was mailed May 19, 2004. An Amendment After Final Rejection was filed July 14, 2004 canceling claim 23 and adding claim 26. An Advisory Action was mailed August 24, 2004. All Amendments are made of record.

A Notice of Appeal and appeal fee, along with a Petition for one-month extension of time were mailed on September 2, 2004.

STATUS OF AMENDMENTS

All the amendments added during prosecution of the application have been entered. The rejected claims 1-7 and 9-22, and 24-26 as they presently stand, are set forth in the attached Appendix. A summary of the rejection of the claims may be found in the Amendment After Final Rejection mailed July 14, 2004 and received by the United States Patent and Trademark Office on September 7, 2004.

SUMMARY OF THE INVENTION

The invention of independent claim 1 is directed to a method for updating existing code in a computer program after inputting new code, which defines changes to the existing code. The method includes generating both a target file list of target files to update and a dependency file list of files dependent on such target files. Specification,

page 9, lines 3-8; page 11, lines 1-6 and Fig. 1, steps 12 and 14. The dependency file list of files are read into a control file, wherein selected lines of the files are split into target strings having programming language substitutions, which are appended to a requisition list, and into prerequisite strings, which are stored in corresponding requisite arrays. Specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1, step 16 and Fig. 2, steps 52-72. An algorithm is then executed wherein the algorithm first matches the target files with the substituted target string in the requisition list in the control file. The algorithm then updates those matched target files if it is determined that the corresponding prerequisite strings stored in the corresponding requisite arrays in the control file have been updated more recently than the substituted target string. Specification, page 9, lines 13-15; page 10, lines 10-26; page 12, line 14 to page 18, line 4; Fig. 1, step 18 and Fig. 3, steps 100-122.

The invention recited in independent claim 5 is directed to a method for generating changes and updating existing files and code in a computer program. The method includes reading existing target files and dependency files in the computer program (specification, page 9, lines 3-8; page 11, lines 1-6 and Fig. 1, steps 12 and 14), followed by reading a plurality of the dependency files associated with the target files into a single control file, wherein selected lines of the dependency files are split into target strings and prerequisite strings (specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1, step 16 and Fig. 2, steps 52-72). As discussed and supported in the specification at page 9, lines 13-15; page 10, lines 10-26; page 12,

line 14 to page 18, line 4; Fig. 1, step 18 and Fig. 3, steps 100-122, a utility program is executed that updates the target files and the dependency files associated with the target files, whereby this utility program includes an interpreted scripting language specifying particular characters to search for in the target files and the associated dependency files. Using this utility program, a requisition list of target strings is then generated which has interpreted scripting language substitutions and corresponding requisite arrays for the prerequisite strings. The target files are then updated by employing a search technique defined in the utility program that includes specified target patterns such that the specified target patterns identify the existing target files being updated, whereby the existing target files are updated if it is determined from the specified target patterns that the prerequisite strings in the control file have been updated more recently than the substituted target string.

The invention disclosed in independent claim 14 is directed to a computer program product for updating existing code in a computer program after inputting new code that defines changes to the existing code. The computer program product has computer readable program code means for generating a target file list of target files to update and a dependency file list of files dependent on the target files. Specification, page 9, lines 3-8; page 11, lines 1-6; page 19, lines 5-19 and Fig. 1, steps 12 and 14. The computer program product also has computer readable program code means for reading the dependency file list of files into a control file, wherein selected lines of the files are split into target strings having programming language substitutions and being appended to a requisition list and

into prerequisite strings being stored in corresponding requisite arrays. Specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1, step 16; page 19, lines 5-19 and Fig. 2, steps 52-72. A computer readable program code means is provided for executing an algorithm where the algorithm matches the target files with the substituted target string in the requisition list in the control file, and then the algorithm updating the target files if it is determined that the corresponding prerequisite strings stored in the corresponding requisite arrays in the control file have been updated more recently than the substituted target string. Specification, page 9, lines 13-15; page 10, lines 10-26; page 12, line 14 to page 18, line 4; page 19, lines 5-19; Fig. 1, step 18 and Fig. 3, steps 100-122.

The invention claimed in independent claim 15 is directed to a program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for updating existing code in a computer program after inputting new code that defines changes to the existing code. The method steps include generating a target file list of target files to update and a dependency file list of files dependent on the target files. Specification, page 9, lines 3-8; page 11, lines 1-6; page 19, lines 5-19 and Fig. 1, steps 12 and 14. The dependency file list of files is read into a control file, wherein selected lines of the files are split into target strings having programming language substitutions and being appended to a requisition list and into prerequisite strings being stored in corresponding requisite arrays. Specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1,

step 16; page 19, lines 5-19 and Fig. 2, steps 52-72. An algorithm is executed wherein the algorithm matches the target files with the substituted target string in the requisition list in the control file, and then the algorithm updates the target files if it is determined that the corresponding prerequisite strings stored in the corresponding requisite arrays in the control file have been updated more recently than the substituted target string. Specification, page 9, lines 13-15; page 10, lines 10-26; page 12, line 14 to page 18, line 4; page 19, lines 5-19; Fig. 1, step 18 and Fig. 3, steps 100-122

The invention of independent claim 16 is directed to a method for updating target files in a computer that includes generating a target file list of target files to update. Specification, page 9, lines 3-8; page 11, lines 1-6; and Fig. 1, steps 12 and 14. A list of files dependent on the target files is read into a control file and selected lines of the dependent files are split into target strings and prerequisite strings, whereby programming language substitutions are performed in the target strings, substituted target strings are appended to a requisition list and the prerequisite strings are stored in corresponding requisite arrays. Specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1, step 16; and Fig. 2, steps 52-72. The method further includes executing an algorithm to match selected target files from the target file list to the substituted target string in the requisition list, retrieving prerequisite strings from the corresponding requisite arrays, and updating the same by performing all possible programming language substitutions to the prerequisite strings using the algorithm. Those prerequisite strings that have been updated more recently than the substituted target string

are identified to generate update rules using the algorithm, whereby the target files from the target file list are then updated using such update rules. Specification, page 9, lines 13-15; page 10, lines 10-26; page 12, line 14 to page 18, line 4; Fig. 1, step 18 and Fig. 3, steps 100-122.

The invention disclosed in independent claim 26 is directed to a method for updating target files in a computer that includes generating a target file list of target files to update. Specification, page 9, lines 3-8; page 11, lines 1-6; and Fig. 1, steps 12 and 14. A list of files dependent on the target files is read into a control file and selected lines of the dependent files are split into target strings and prerequisite strings, whereby programming language substitutions are performed in the target strings, substituted target strings are appended to a requisition list and the prerequisite strings are stored in corresponding requisite arrays. Specification, page 9, lines 9-12; page 9, line 16 to page 10, line 9; page 11, line 7 to page 12, line 13; Fig. 1, step 16; and Fig. 2, steps 52-72. An algorithm is executed to match selected target files from the target file list to the substituted target string in the requisition list, retrieve prerequisite strings from the corresponding requisite arrays, and update the same by performing all possible programming language substitutions to the prerequisite strings using the algorithm. Those prerequisite strings that have been updated more recently than the substituted target string are identified to generate update rules using the algorithm, whereby these update rules support multi-directory builds from a single control file (*emphasis*, specification, page 13, line 21 to page 14, line 21). The target files from the target file list are then updated using such update rules. Specification, page 9,

lines 13-15; page 10, lines 10-26; page 12, line 14 to page 18, line 4; Fig. 1, step 18 and Fig. 3, steps 100-122.

ISSUES

1. According to the substantive rejections set forth in the above Final Rejection, claims 1-7, 9-11, 14-19 and 23-25 (i.e., not claims 1-11, 13-19 and 23-25 recited in paragraph 6 of such Final Rejection) stand rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000).

2. According to the substantive rejections set forth in the above Final Rejection, claims 12, 13 and 20-22 (i.e., not claims 12 and 20-23 recited in paragraph 7 of such Final Rejection) stand rejected under 35 USC 103(a) as being unpatentable over The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000) in view of Welsh, "Practical Programming in Tcl and Tk", (1999).

ARGUMENT

I. Rejection under 35 USC 102(a) over The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000).

Claims 1-4

As discussed above, independent claim 1 is directed to a method for updating existing code in a computer program after inputting new code that defines changes to the existing code. The method includes generating a target file list of target files to update,

and a dependency file list of files dependent on the target files. The dependency file list is read into a control file, and once therein, selected lines of these files are split into target strings and prerequisite strings. Those target strings having programming language substitutions are appended to a requisition list, while the prerequisite strings are stored in corresponding requisite arrays thereto. An algorithm is executed, whereby this algorithm first matches target files from the target file list with the substituted target string in the requisition list in the control file. The algorithm then updates those matched target files from the target file list if it is determined that the corresponding prerequisite strings, which are stored in the control file in the corresponding requisite arrays, have been updated more recently than the substituted target string.

Appellant submits that The GNU Make Manual is directed to a program that uses a conventional “*make*” utility, at which the present invention is aimed at improving upon and overcoming the problems associated therewith. (See, Specification, p. 1, l. 8 – p. 4, l. 25.) As recited in appellant’s application, the present invention provides improvements to conventional *make* utilities that use descriptor files containing dependency rules, macros, and suffix rules to instruct *make* to automatically rebuild the program whenever one of the program component files is modified. The *make* utility operates by following rules that are provided in its descriptor file, typically called *makefile*. (Specification, p. 1, ll. 20-23.)

The GNU Make Manual discloses a *make* utility that determines which pieces of a large program needs to be recompiled, and issues the commands to recompile them. (GNU MM, p. 1, ll. 23-25 and p. 6., ll. 3-5.) To prepare to use *make*, one must write a

makefile that describes the relationships among files in the program and provides commands for updating each file. (GNU MM, p. 6., ll. 20-22.) Once a suitable *makefile* exists, each time a source file is changed, a simple shell command performs recompilations. (GNU MM, p. 6., ll. 26-28.) The *make* program uses the *makefile* data base and the last modification times of the fields to decide which of the files need to be updated, and then issues the command recorded in the data base. (GNU MM, p. 6., ll. 28-31.)

The GNU *makefile* contains five kinds of things: explicit rules, implicit rules, variable definitions, directives, and comments, whereby *make* works in two distinct phases. (GNU MM, p. 15, l. 28-30 and p.21, l. 41.) During the first phase, *make* reads all the *makefiles*, and internalizes all the variables and their values, implicit and explicit rules, and constructs a dependency graph of all the targets and their prerequisites. (GNU MM, p.21, ll. 41-45.) During the second phase, *make* uses these internal structures to determine what targets will need to be rebuilt and to invoke the rules necessary to do so. (GNU MM, p.21, l. 45-47.) When the objects of a *makefile* are created only by implicit rules, entries may be grouped by their prerequisites instead of by their targets. (GNU MM, p.14, ll. 9-11.) The commands of a rule consist of shell command lines to be executed one by one. (GNU MM, p. 47, ll. 35-36.) Users use many different shell programs, but commands in *makefiles* are always interpreted by */bin/sh* unless the *makefile* specifies otherwise. (GNU MM, p. 47, ll. 43-45.)

Page 9, lines 35-50 of The GNU Make Manual show a straightforward *makefile* that describes the way an executable file called *edit* depends on eight object files, which in turn depend on eight C source and three header files. (GNU MM, p. 9, ll. 29-31.) In the example *makefile*, the targets include the executable file *edit*, and the object files "*main.o*" and "*kbd.o*", whereby each ".o" file is both a target and a prerequisite. When a target is a file, it needs to be recompiled or relinked if any of its prerequisites change. In addition, any prerequisites that are themselves automatically generated should be updated first. (GNU MM, p. 10, ll. 21-28.) A shell command follows each line that contains a target and prerequisites, whereby these shell commands say how to update the target file. (GNU MM, p. 10, ll. 33-35.) A tab character must come at the beginning of every command line to distinguish commands lines from other lines in the *makefile*. (Bear in mind that *make* does not know anything about how the commands work. It is up to you to supply commands that will update the target file properly. All *make* does is execute the commands in the rule you have specified when the target file needs to be updated.) (GNU MM, p. 10, ll. 35-40.)

Thus, as recited in the foregoing application and in accordance with a conventional "*make*" utility, the description of the "*make*" utility in The GNU Make Manual requires multiple *makefiles* to specify cross-directory file dependencies. (Specification, p. 3, ll. 12-13.) Also, the *make* command requires building a substantially complete dependency tree before it starts, which can take an extensive amount of time and computer resources. (Specification, p. 2, ll. 13-17.)

Appellant submits that the present invention is not anticipated by The GNU Make Manual. A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference. *Verdegaal Bros. Inc. v. Union Oil Co.*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1052 (Fed. Cir.), *cert. denied*, 484 U.S. 827 (1987). The inquiry as to whether a reference anticipates a claim must focus on what subject matter is encompassed by the claim and what subject matter is described in the reference. As set forth by the court in *Kalman v. Kimberly-Clark Corp.* 713 F.2d 760, 772, 218 USPQ 781, 789 (Fed. Cir. 1983), *cert. denied*, 465 U.S. 1026 (1984), it is only necessary for the claims to “ ‘read on’ something disclosed in the reference, i.e., all limitations of the claim are found in the reference, or ‘fully met’ by it.” It is also well known that the prior art may anticipate when complete anticipation requires only that one skilled in the art merely exercise that skill to complete the work and practice the patented invention without having to depend on other prior art or the inventor’s own inventive skills. *Studiengesellschaft Kohle mbH v., Dart Industries, Inc.* (CAFC 1984).

With respect to the Examiner’s arguments, The GNU Make Manual does not disclose or even contemplate reading dependency files into a control file, and once therein, splitting selected lines of these dependency files into target strings and prerequisite strings, as is currently claimed. The Examiner cites page 9, lines 35-50, of The GNU Make Manual, which merely shows a straightforward *makefile*. (GNU MM, p. 9, ll. 29-31.) As disclosed in The GNU Make Manual, a shell command follows each line that contains a target and prerequisites, whereby these shell commands say how to update the target file.

(GNU MM, p. 10, ll. 33-35); however, “bear in mind that *make* does not know anything about how the commands work. It is up to you to supply commands that will update the target file properly. All make does is execute the commands in the rule you have specified when the target file needs to be updated” (GNU MM, p. 10, ll. 35-40).

Unlike appellant’s invention, it is submitted that The GNU Make Manual does not disclose reading dependency files into a control file. Since The GNU Make Manual does not disclose reading dependency files into a control file it cannot, and does not, disclose updating split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string. The GNU Make Manual does not disclose performing such a step by executing an algorithm that matches target files from the target file list with the substituted target string in the requisition list in the control file if it is determined that the corresponding prerequisite strings, which are stored in and retrieved from the corresponding requisite arrays in the control file, have been updated more recently than the substituted target string. Further, The GNU Make Manual does not recite appending target strings having programming language substitutions to a requisition list while the prerequisite strings are stored in corresponding requisite arrays thereto.

Appellant submits that a critical feature of applicant’s invention is that a plurality of dependency files associated with target files are read into a control file, and split into target and prerequisite strings, whereby those target strings having programming language substitutions are appended to a requisition list while the prerequisite strings are appended

to corresponding requisite arrays. Using an algorithm that includes specified target patterns, the target files are updated if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string. The GNU Make Manual does not disclose any of the foregoing.

Moreover, the Final Rejection dated May 19, 2004 the Examiner bases his rejection of claims 1-4 as being anticipated by The GNU Make Manual under 35 USC 102(a). However, in the Advisory Action dated August 24, 2004 the Examiner states that "the GNU MM grouping of the entries by their prerequisites allows for the identical operations/functions to be performed on the prerequisites and target strings as in the situation where the target and prerequisite strings are separated Since, identical functions are capable of being performed, the separation of the dependency files into target and prerequisite strings and the use of a single control file are a matter of design choice and do not distinguish appellant's application over the prior art." However, appellant submits that, in accordance with case law in this area, a matter of "design choice" argument is a basis for an obviousness rejection under 35 USC 103 (i.e., "obvious design choice"), not an anticipation rejection under 35 USC 102. See, *In re Gal*, 980 F.2d 717, 25 USPQ2d 1076 (Fed. Cir. 1992) (finding of "obvious design choice" precluded where the claimed structure and the function it performs are different from the prior art).

Accordingly, in view of the foregoing arguments and the Examiner's comments in the above Advisory Action, it is apparent that the Examiner recognizes that The GNU Make Manual does not disclose, or even contemplate, the use of a control file (i.e., a

single control file), such that claims 1-4 contain limitations not disclosed or contemplated by The GNU Make Manual reference. It is for these reasons that Appellant submits that The GNU Make Manual does not anticipate independent claim 1, and claims 2-4 dependent thereon.

Claims 5-7 and 9-11

Claims 5-7 and 9-11 have been rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual. Appellant disagrees and herein incorporates and reasserts the above arguments made with respect to claims 1-4 for pending claims 5-7 and 9-11, as well as submits the below remarks.

Independent claim 5 is also directed to a method for generating changes and updating existing files and code in a computer program by first reading existing target files and dependency files in the computer program. A plurality of these dependency files associated with the target files are then read into a single control file. Once in the single control file, selected lines of the dependency files are split into target strings and prerequisite strings. A utility program is executed for updating the target files and associated dependency files, whereby the utility program includes an interpreted scripting language specifying particular characters to search for in the target files and associated dependency files. Using the utility program, a requisition list of target strings having interpreted scripting language substitutions is then generated, as well as corresponding requisite arrays for the prerequisite strings. The target files are then updated by employing a search technique defined in the utility program that includes specified target patterns

identifying the existing target files being updated. The existing target files are updated if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string.

In addition to the above remarks and arguments, Appellant submits that The GNU Make Manual does not disclose or even contemplate reading a plurality of dependency files associated with target files into a single control file, and once therein, splitting selected lines of these dependency files into target strings and prerequisite strings, as is currently claimed. Rather, the pages of The GNU Make Manual cited by the Examiner (GNU MM, p.14, ll. 9-11) merely disclose that “[w]hen the objects of a *makefile* are created only by implicit rules, entries may be grouped by their prerequisites instead of by their targets.

The GNU Make Manual also does not recite performing programming language substitutions in these target strings using an interpreted scripting language, whereby target strings having programming language substitutions are appended to a requisition list while the prerequisite strings are stored in corresponding requisite arrays thereto. The GNU Make Manual merely discloses that the commands of a rule consist of shell command lines to be executed one by one. (GNU MM, p. 47, ll. 35-36.) Users use many different shell programs, but commands in *makefiles* are always interpreted by `/bin/sh` unless the *makefile* specifies otherwise. (GNU MM, p. 47, ll. 43-45.) This is not an interpreted scripting language. As one skilled in the art will understand, there are two parts to an update rule: the conditional (when something needs to be done) and the action (what

might need to be done). The "interpretation" referred to in The GNU Make Manual refers to the interpretation of the shell commands which are part of the "action". Appellant's claimed "interpreted scripting language" is part of the system for processing the 'conditional' part of the rules. This distinction is central to the present invention because the use of interpretation in processing the rules themselves provides critical support for multi-directory builds from a single control file, and is considerably more universal and powerful than simple textual substitution.

Further, it is submitted that one skilled in the art will appreciate that The GNU Make Manual's `gmake vpath/VPATH` has no way of processing multiple source files having the same name but which are located in different source subdirectories, unless multiple `gmake` control files are used. Additionally, consider the problem of using `gmake` to copy all recently changed files in a source directory tree `SRCDIR` to a target directory tree `TRGDIR`. Using `gmake`, one would have to write a separate control file for each subdirectory in the `SRCDIR` subtree, because `vpath/VPATH` is inadequate for this application, and cannot be extended to handle it. The present invention enables one to generate update rules that support multi-directory builds from a single control file, as is currently claimed.

As recognized by the Examiner in the Advisory Action dated August 24, 2004, The GNU Make Manual does not disclose or contemplate the use of a single control file, nor does it disclose using an interpreted scripting language. Accordingly, Appellant submits that claims 5-7 and 9-11 include limitations not disclosed or contemplated by The GNU

Make Manual reference, such that, these claims are not anticipated by The GNU Make Manual.

Claim 14

Claim 14 has been rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual. Appellant disagrees and herein incorporates and reasserts the above arguments made with respect to claims 1-7 and 9-11 for pending claim 14, as well as submits the below remarks.

Independent claim 14 is directed to a computer program product for updating existing code in a computer program after inputting new code that defines changes to the existing code. The computer program product has computer readable program code means for generating a target file list of target files to update and a dependency file list of files dependent on the target files. The computer program product also has computer readable program code means for reading the dependency file list of files into a control file, wherein selected lines of the files are split into target strings having programming language substitutions and being appended to a requisition list and into prerequisite strings being stored in corresponding requisite arrays. A computer readable program code means is provided for executing an algorithm where the algorithm matches the target files with the substituted target string in the requisition list in the control file, and then the algorithm updating the target files if it is determined that the corresponding prerequisite strings stored in the corresponding requisite arrays in the control file have been updated more recently than the substituted target string.

Unlike the limitations recited in independent claim 14, and as discussed in detail above, The GNU Make Manual does not disclose reading dependency files into a control file, and as such, it does not disclose updating split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string. Hence, The GNU Make Manual does not disclose performing such a step by executing an algorithm that matches target files from the target file list with the substituted target string in the requisition list in the control file if it is determined that the corresponding prerequisite strings, which are stored in and retrieved from the corresponding requisite arrays in the control file, have been updated more recently than the substituted target string. Further, The GNU Make Manual does not recite appending target strings having programming language substitutions to a requisition list while the prerequisite strings are stored in corresponding requisite arrays thereto.

Again, as recognized by the Examiner in the Advisory Action dated August 24, 2004, The GNU Make Manual does not disclose or contemplate the use of a control file, and as such, claim 14 contains limitations not disclosed or contemplated by The GNU Make Manual reference such that claim 14 is not anticipated by The GNU Make Manual.

Claim 15

Claim 15 has been rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual. Appellant disagrees and herein incorporates and reasserts the above arguments made with respect to claims 1-7, 9-11 and 14 for pending claim 15, as well as submits the below remarks.

Independent claim 15 is directed to a program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for updating existing code in a computer program after inputting new code that defines changes to the existing code. The method steps include generating a target file list of target files to update and a dependency file list of files dependent on the target files. The dependency file list of files is read into a control file, wherein selected lines of the files are split into target strings having programming language substitutions and being appended to a requisition list and into prerequisite strings being stored in corresponding requisite arrays. An algorithm is executed wherein the algorithm matches the target files with the substituted target string in the requisition list in the control file, and then the algorithm updates the target files if it is determined that the corresponding prerequisite strings stored in the corresponding requisite arrays in the control file have been updated more recently than the substituted target string.

As discussed above, The GNU Make Manual does not disclose reading dependency files into a control file (as recognized by the Examiner in the Advisory Action dated August 24, 2004). As such, The GNU Make Manual does not disclose updating split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string. The GNU Make Manual also does not disclose performing such a step by executing an algorithm that matches target files from the target file list with the substituted target string in the requisition list in the control file if it is determined that the corresponding prerequisite strings, which are stored in and retrieved

from the corresponding requisite arrays in the control file, have been updated more recently than the substituted target string.

It is for these reasons that Appellant submits that claim 15 includes limitations not disclosed or contemplated by The GNU Make Manual such that claim 15 is not anticipated by The GNU Make Manual.

Claims 16-19 and 24-25

Claims 16-19 and 24-25 have been rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual. Appellant disagrees and herein incorporates and reasserts the above arguments made with respect to claims 1-7, 9-11, 14 and 15 for pending claims 16-19 and 24-25, as well as submits the below remarks.

Independent claim 16 is directed to a method for updating target files in a computer by generating a target file list of target files to update and reading into a control file a list of files dependent on the target files. Selected lines of these dependent files are split into target strings and prerequisite strings, and then programming language substitutions are performed in the target strings. The substituted target strings are appended to a requisition list, which is stored in corresponding requisite arrays. An algorithm is then executed to match selected target files from the target file list to the substituted target string in the requisition list. The prerequisite strings are then retrieved from the corresponding requisite arrays and updated by performing all possible programming language substitutions using the algorithm. Using the algorithm, those prerequisite strings that have been updated

more recently than the substituted target string are identified to generate update rules, and then the target files from the target file list are updated using such update rules.

Appellant submits that The GNU Make Manual does not disclose reading dependency files into a control file, and as such, it does not disclose updating split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string, as is currently claimed in independent claim 16. As such, The GNU Make Manual does not disclose performing such a step by executing an algorithm that matches target files from the target file list with the substituted target string in the requisition list in the control file if it is determined that the corresponding prerequisite strings, which are stored in and retrieved from the corresponding requisite arrays in the control file, have been updated more recently than the substituted target string. Further, The GNU Make Manual does not recite appending target strings having programming language substitutions to a requisition list while the prerequisite strings are stored in corresponding requisite arrays thereto.

Further, the “directory searching” in The GNU Make Manual’s *gmake* is not dynamic directory switching to specify multiple files in multiple directories as is recited in claim 24. Applicant’s dynamic directory switching allows the user to conveniently switch from one directory to another as he or she specifies prerequisites. (See, specification, p. 14, l. 26 to p. 15, l. 7.) With respect to claim 25, The GNU Make Manual does not disclose that an algorithm considers a directory to be out-of-date regardless of its time stamp (i.e., always out-of-date regardless of its time stamp) such that any rule associated

with directory target is always triggered. The pages cited by the Examiner, p. 120 line 5 - p. 127 line 50 "(pattern matching rules and automatic variables are used to specify rules for directories)" do not support the recited limitations of claim 25.

Accordingly, claims 16-19 and 24-25 include limitations not disclosed or contemplated by The GNU Make Manual, such that The GNU Make Manual does not render obvious claims 16-19 and 24-25.

Claim 26

Claim 26 has been rejected under 35 USC 102(a) as being anticipated by The GNU Make Manual. Appellant disagrees and herein incorporates and reasserts the arguments made with respect to claims 1-7, 9-11, 14-19, 24 and 25 above, for pending claim 26, as well as submits the below remarks.

Independent claim 26, which includes the limitations of base claim 23 and dependent claim 16, is directed to a method for updating target files in a computer that includes generating a target file list of target files to update. A list of files dependent on the target files is read into a control file and selected lines of the dependent files are split into target strings and prerequisite strings, whereby programming language substitutions are performed in the target strings, substituted target strings are appended to a requisition list and the prerequisite strings are stored in corresponding requisite arrays. An algorithm is executed to match selected target files from the target file list to the substituted target string in the requisition list, retrieve prerequisite strings from the corresponding requisite arrays, and update the same by performing all possible programming language substitutions to the

prerequisite strings using the algorithm. Those prerequisite strings that have been updated more recently than the substituted target string are identified to generate update rules using the algorithm, whereby these update rules support multi-directory builds from a single control file. The target files from the target file list are then updated using such update rules.

Unlike the limitations recited in independent claim 26, The GNU Make Manual does not disclose or contemplate reading a plurality of dependency files associated with target files into a single control file, and once therein, splitting selected lines of these dependency files into target strings and prerequisite strings, as is currently claimed. As is currently claimed, update rules are generated using the algorithm by identifying those prerequisite strings that have been updated more recently than the substituted target string. These update rules support multi-directory builds from a single control file. Applicant submits that The GNU Make Manual's Gmake's vpath/VPATH facility does not disclose multi-directory builds from a single control file. It is a conventional *make* utility that uses conventional *makefiles*, whereby multiple *makefiles* are needed to specify cross-directory file dependencies. (Specification, p. 2, l. 6 to p. 3 l. 13.) Further, the passages of The GNU Make Manual cited by the Examiner, namely, "The directory search features of make facilitate this by searching several directories automatically to find a prerequisite" at page 28, lines 51-53, do not disclose update rules that support multi-directory builds from a single control file.

Once again, as recognized by the Examiner in the Advisory Action dated August 24, 2004, The GNU Make Manual does not disclose or contemplate the use of a control file (i.e., single control file), such that it cannot and does not disclose update rules that support multi-directory builds from a single control file as is currently claimed. Claim 26 includes limitations not disclosed or contemplated by The GNU Make Manual reference, such that, claim 26 is not anticipated by The GNU Make Manual.

II. Rejection under 35 USC 103(a) over The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000 in view of Welsh, "Practical Programming in Tcl and Tk", (1999)).

Claims 12-13

Claims 12-13 have been rejected under 35 USC 103(a) as being unpatentable over The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000) in view of Welch, "Practical Programming in Tcl and Tk", (1999).

Appellant herein incorporates and reasserts the arguments made in the above paragraph relating to independent claim 5, from which claims 12 and 13 depend, as well as submits the below remarks.

The GNU Make Manual does not disclose, contemplate or suggest reading a plurality of dependency files associated with target files into a single control file (i.e., a control file) and splitting such dependency files into target strings and prerequisite strings, such that it does not disclose, contemplate or suggest performing programming language substitutions in these target strings, preferably using an interpreted scripting language, such

as, *updt*, *perl*, or *Tcl* (claim 12). As such, The GNU Make Manual does not disclose, contemplate or suggest executing a utility program to update the split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string.

The "Practical Programming in *Tcl* and *Tk*" (1999) reference does not overcome the above deficiencies of The GNU Make Manual as it is merely directed to *Tcl* scripting and is delivered as a practical guide to help users of the *Tcl* and *Tk* programming languages get the most out of *Tcl* and *Tk*. (Welch, p. xivi, ll. 5-9 and 17.) Welch sets up a set of programming techniques that exploit the power of *Tcl* and *Tk* while avoiding troublesome areas. (Welch, p. xivi, ll. 7-8.) It does not disclose, contemplate or suggest reading a plurality of dependency files associated with target files into a control file (i.e., a single control file) and splitting such dependency files into target strings and prerequisite strings, as is currently claimed. As such, Welch does not disclose, contemplate or suggest performing programming language substitutions in these target strings, preferably using an interpreted scripting language, and executing a utility program to update the split target files if it is determined that the prerequisite strings in the control file have been updated more recently than the substituted target string, also as is claimed. Welch does not overcome the above deficiencies of The GNU Make Manual.

Appellant submits that neither The GNU Make Manual nor the Welch reference, taken singly or in any proper combination thereof, discloses the instant invention, such

that, pending claims 12 and 13 are neither anticipated by nor rendered obvious over The GNU Make Manual or the Welch reference, alone or in any proper combination.

Claims 20-22

Claims 20-22 have been rejected under 35 USC 103(a) as being unpatentable over The GNU Make Manual, Version 3.79, edition 0.55, (04/04/2000) in view of Welch, "Practical Programming in Tcl and Tk", (1999).

Appellant herein incorporates and reasserts the above arguments made in the paragraphs relating to both independent claim 16, from which claims 20-22 depend, as well as claims 12 and 13. Since

The GNU Make Manual does not disclose executing an algorithm that matches target files from the target file list with the substituted target string in the requisition list in the control file if it is determined that the corresponding prerequisite strings, which are stored in and retrieved from the corresponding requisite arrays in the control file, have been updated more recently than the substituted target string. That is, those prerequisite strings that have been updated more recently than the substituted target strings are identified to generate update rules, and then the target files from the target file list are updated using such update rules. The update rules are specified using a scripting language selected from *updt*, perl, and Tcl (claim 20), whereby the algorithm is executed in such scripting language (claim 21) and the rules have access to the interpreted programming language to recursively invoke the algorithm on a new target (claim 22). The Welch reference does not overcome the deficiencies of The GNU Make Manual as it

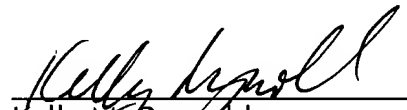
is merely directed to Tcl scripting and is delivered as a practical guide to help users of the Tcl and Tk programming languages get the most out of Tcl and Tk. (Welch, p. xivi, ll. 5-9 and 17.)

Appellant submits that neither The GNU Make Manual nor the Welch reference, taken singly or in any proper combination thereof, discloses the instant invention, such that, pending claims 20-22 are neither anticipated by nor rendered obvious over The GNU Make Manual or the Welch reference, alone or in any proper combination.

SUMMARY

For the reasons given above, Appellant submits that the instant application is in condition for allowance. The final rejection should be reversed and the claims allowed to issue.

Respectfully submitted,


Kelly M. Reynolds
Reg. No. 47,898

DELIO & PETERSON LLC
121 Whitney Avenue
New Haven, CT 06510-1241
(203) 787-0595
ibmf100320000briefapp

APPENDIX

Pending claims of Serial No. 09/757,517.

- 1 1. (Previously presented) A method for updating existing code in a computer
2 program after inputting new code which defines changes to said existing code
3 comprising the steps of:
4 generating a target file list of target files to update;
5 generating a dependency file list of files dependent on said target files;
6 reading said dependency file list of files into a control file, wherein selected
7 lines of said files are split into target strings having programming language
8 substitutions and being appended to a requisition list and into prerequisite
9 strings being stored in corresponding requisite arrays; and
10 executing an algorithm where said algorithm matches said target files with said
11 substituted target string in said requisition list in said control file, and then
12 said algorithm updating those matched target files if it is determined that
13 the corresponding prerequisite strings stored in the corresponding requisite
14 arrays in said control file have been updated more recently than said
15 substituted target string.

- 1 2. (Previously presented) The method of claim 1 further including source code
2 and object code, said target files being source code and said dependency files

3 being object code, said source code being selectively compiled to update said
4 object code.

1 3. (Previously presented) The method of claim 1 further comprising the step
2 of:
3 updating said prerequisite strings with new files defined by new code.

1 4. (Previously presented) The method of claim 2 wherein said algorithm
2 utilizes a search technique including pattern type variables which use generic rules
3 to specify said object code for updating.

1 5. (Previously presented) A method for generating changes and updating
2 existing files and code in a computer program, comprising the steps of:
3 reading existing target files and existing dependency files in said computer
4 program;
5 reading a plurality of said dependency files associated with said target files
6 into a single control file, wherein selected lines of said dependency files are
7 split into target strings and prerequisite strings;
8 executing a utility program which updates said target files and said dependency
9 files associated with said target files, said utility program including an
10 interpreted scripting language specifying particular characters to search for
11 in said target files and said associated dependency files;

12 generating a requisition list of target strings having interpreted scripting
13 language substitutions and corresponding requisite arrays for said
14 prerequisite strings using said utility program; and
15 updating said target files by employing a search technique defined in said
16 utility program, said search technique includes specified target patterns
17 such that said specified target patterns identify said existing target files being
18 updated, said existing target files being updated if it is determined from said
19 specified target patterns that said prerequisite strings in said control file
20 have been updated more recently than said substituted target string.

1 6. (Previously presented) The method of claim 5 wherein said specified target
2 patterns of said search technique includes pattern type variables which use generic
3 rules to specify said target files for updating.

1 7. (Previously presented) The method of claim 5 wherein said search
2 technique includes matching specified characters in said target files to said
3 requisition list of target strings such that said specified characters identify said
4 existing target files being updated.

1 8. (Canceled.)

1 9. (Previously presented) The method of claim 5 wherein said utility program
2 defines new target files to be added to said existing target files.

1 10. (Previously presented) The method of claim 5 wherein the utility program
2 prioritizes said target files to update while employing said search technique.

1 11. (Previously presented) The method of claim 5 wherein said utility program
2 includes a process procedure for an operator to call, said process procedure
3 recursively invokes said utility program and arguments.

1 12. (Previously presented) The method of claim 5 wherein said utility program
2 is in a scripting language selected from the group consisting of *updt*, *perl*, and *Tcl*.

1 13. (Previously presented) The method of claim 5 wherein said utility program
2 provides that said existing target files with a specific character are not considered
3 to be a file, and thereby are bypassed for any changes by the utility program.

1 14. (Previously presented) A computer program product for updating existing
2 code in a computer program after inputting new code which defines changes to
3 said existing code, said computer program product having:

4 computer readable program code means for generating a target file list of target
5 files to update;

6 computer readable program code means for generating a dependency file list
7 of files dependent on said target files;

8 computer readable program code means for reading said dependency file list of
9 files into a control file, wherein selected lines of said files are split into
10 target strings having programming language substitutions and being
11 appended to a requisition list and into prerequisite strings being stored in
12 corresponding requisite arrays; and

13 computer readable program code means for executing an algorithm where said
14 algorithm matches said target files with said substituted target string in said
15 requisition list in said control file, and then said algorithm updating said
16 target files if it is determined that the corresponding prerequisite strings
17 stored in the corresponding requisite arrays in said control file have been
18 updated more recently than said substituted target string.

1 15. (Previously presented) A program storage device readable by a machine,
2 tangibly embodying a program of instructions executable by the machine to
3 perform method steps for updating existing code in a computer program after
4 inputting new code which defines changes to said existing code, said method steps
5 comprising:

6 generating a target file list of target files to update;
7 generating a dependency file list of files dependent on said target files;
8 reading said dependency file list of files into a control file, wherein selected
9 lines of said files are split into target strings having programming language
10 substitutions and being appended to a requisition list and into prerequisite
11 strings being stored in corresponding requisite arrays; and

12 executing an algorithm where said algorithm matches said target files with said
13 substituted target string in said requisition list in said control file, and then
14 said algorithm updating said target files if it is determined that the
15 corresponding prerequisite strings stored in the corresponding requisite
16 arrays in said control file have been updated more recently than said
17 substituted target string.

1 16.(Previously presented) A method for updating target files in a computer
2 comprising:
3 generating a target file list of target files to update;
4 reading into a control file a list of files dependent on said target files;
5 splitting selected lines of said dependent files into target strings and
6 prerequisite strings;
7 performing programming language substitutions in said target strings;
8 appending said substituted target strings to a requisition list;
9 storing said prerequisite strings in corresponding requisite arrays;
10 executing an algorithm to match selected target files from said target file list to
11 said substituted target string in said requisition list;
12 retrieving said prerequisite strings from said corresponding requisite arrays;
13 updating said prerequisite strings by performing all possible programming
14 language substitutions to said prerequisite strings using said algorithm;

15 identifying those prerequisite strings that have been updated more recently
16 than said substituted target string to generate update rules using said
17 algorithm; and
18 updating said target files from said target file list using said update rules.

1 17. (Previously presented) The method of claim 16 wherein after said list of files
2 dependent on said target files are read into said control file, said remaining
3 subsequent steps utilize said control file.

1 18. (Previously presented) The method of claim 16 further including updating
2 said target file list with new target files, the new target files being defined by said
3 update rules

1 19. (Previously presented) The method of claim 16 wherein said update rules
2 comprise target patterns to specify entire classes of dependencies.

1 20. (Previously presented) The method of claim 16 wherein the update rules are
2 specified using a scripting language selected from the group consisting of *updt*,
3 perl, and Tcl.

1 21. (Previously presented) The method of claim 20 wherein said algorithm is
2 executed in said scripting language.